Le système graphique X Window

Introduction

Historique

- développé au MIT en 1984 (projet ATHENA)
- dérivé du système de fenêtrage W (Stanford)
- la version la plus connue : version 11 (X11) - 1987
- dernière release : X11R6 – 1994

Caractéristiques

- gestion de l'affichage en mode graphique
  - fenêtres, dessins, texte, images
  - gestion des entrées (clavier, souris)
- couche au dessus du système d'exploitation
  - indépendante du matériel et du système d'exploitation
- utilisation en réseau transparente et optimisée

Concepts de base

Les fenêtres

- contenu : dessin graphique, texte, images
- gestion du chevauchement
- association des entrées (clavier, souris) aux fenêtres

Le modèle client-serveur

- un serveur graphique et plusieurs clients (applis)
- communication bi-directionnelle

Le fonctionnement en réseau (protocole X)

- communication entre les clients et le serveur
- utilisation des couches réseau de transport (ex. TCP, DECNet)

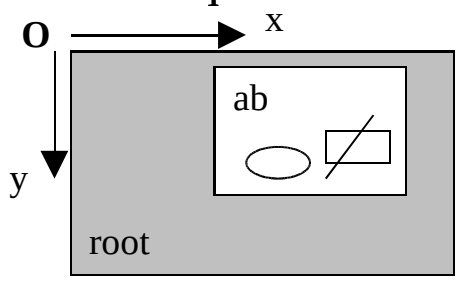
Librairies d'interface au protocole X

- la plus connue est Xlib (langage C)
- permettent l'écriture d'applications X Window

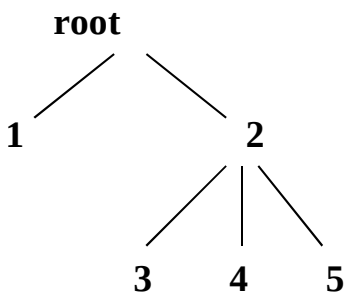
Les fenêtres

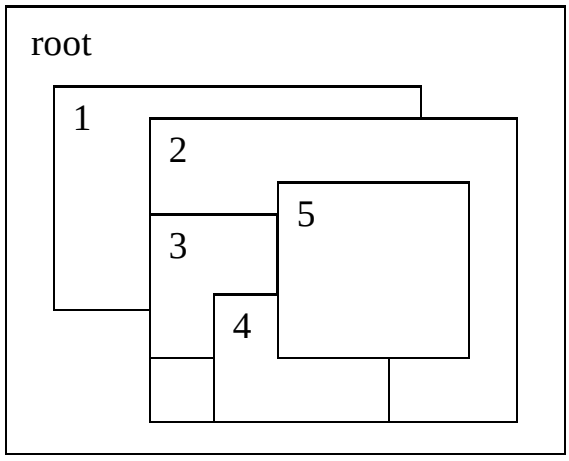
Caractéristiques

- zone rectangulaire composée de *pixels*
- primitives de dessin qui modifient ces pixels

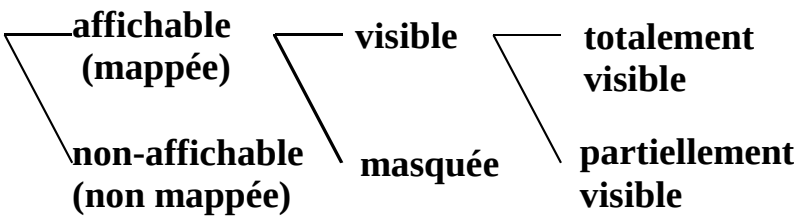


Structure arborescente





Etat d'une fenêtre



Architecture client - serveur

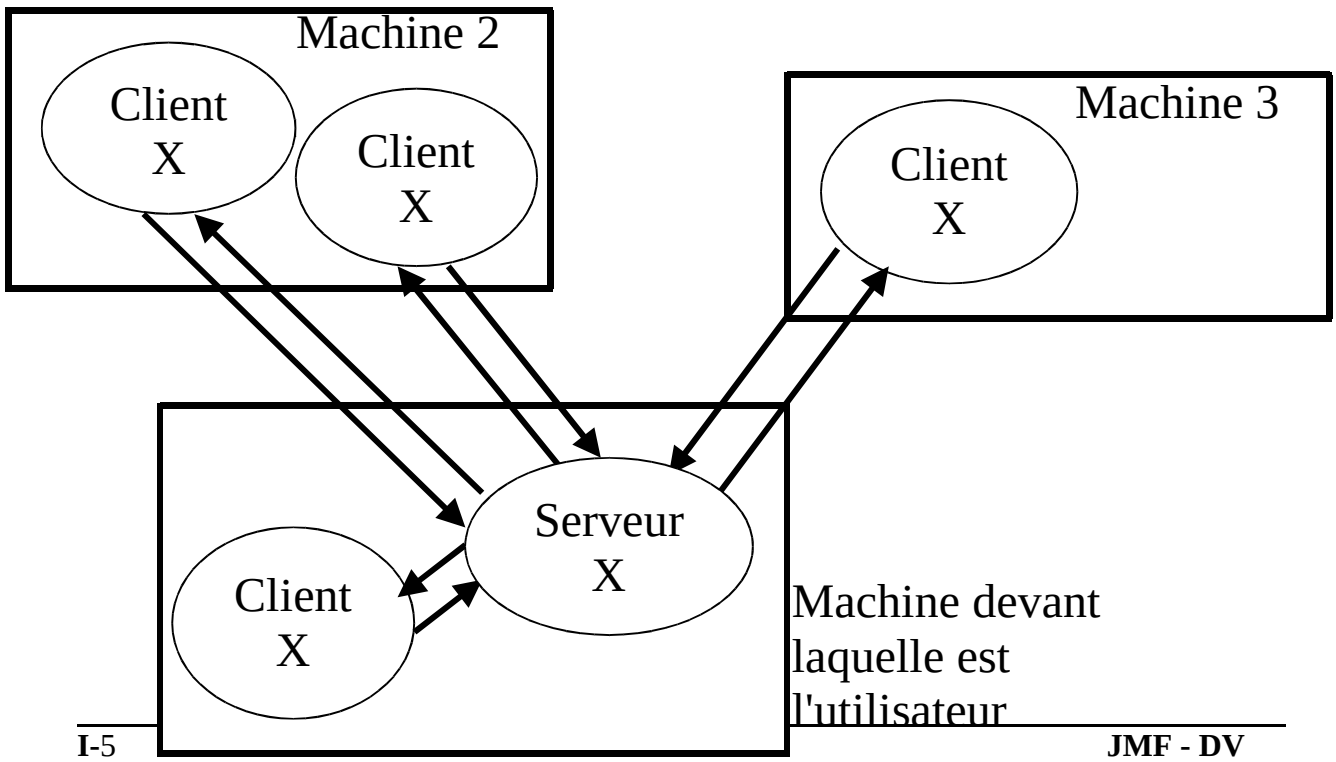
Comme d'habitude, un serveur rend un service, un client de ce service est un demandeur de ce service. Ici le service est un service d'interface homme machine.

Signification particulière des termes

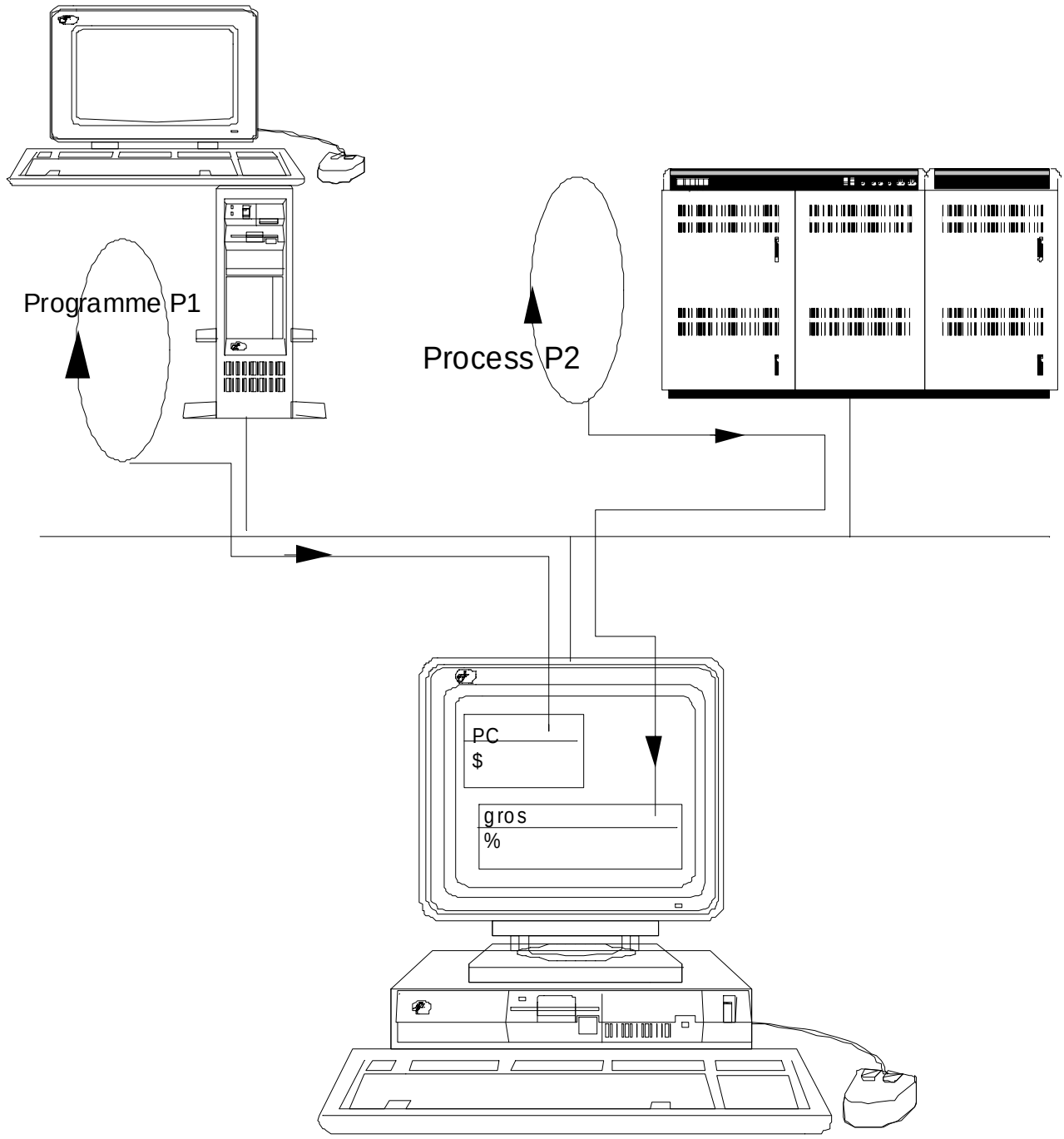
serveur X : programme qui gère l'écran d'affichage, le clavier et la souris

client X : programme applicatif qui peut s'exécuter sur la même machine que le serveur X ou ailleurs

Remarque : en bases de données le client est « ici » (sur le poste de travail) et le serveur est « ailleurs ». En X Window la localisation est inversée



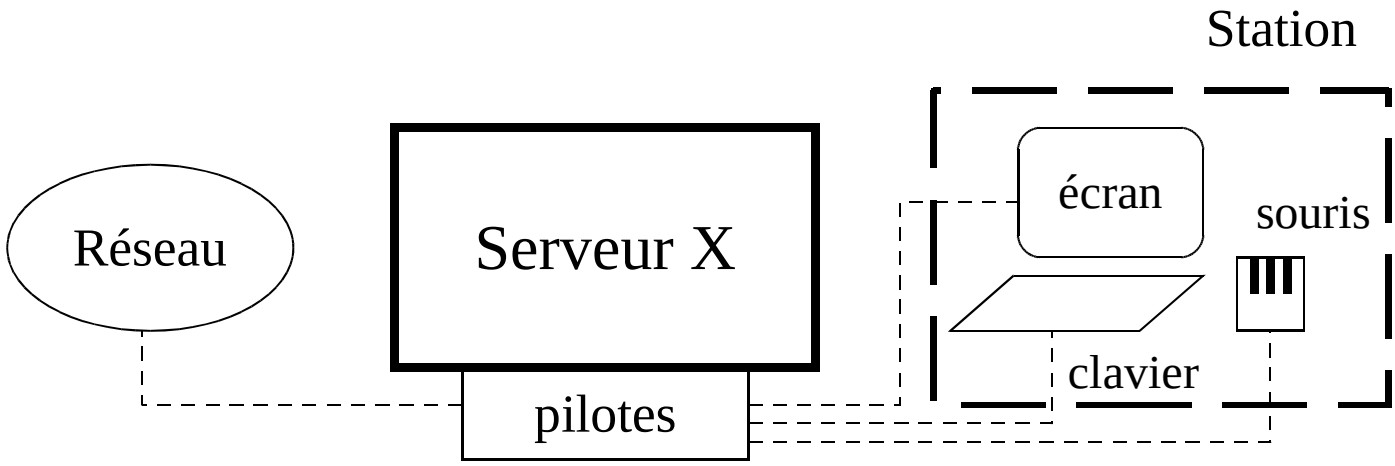
Architecture X Window



Le serveur X

Composantes

- pilote d'écran
- pilotes de clavier et de souris
- pilote de communication (en réseau)



Display et écrans

- un display correspond à un serveur X ou au canal de communication qui mène à ce serveur X

  plusieurs display possibles sur une machine

- un display peut gérer plusieurs écrans
  adressage : nom_host (adresse IP): n°display.n° écran

  Ex. fermi:0.0,
  192.70.50.2:0.1

- en général un seul display (n° 0) et un écran (n° 0)

Clients

Les clients

- programmes dont l'affichage et les entrées sont gérés par un serveur X

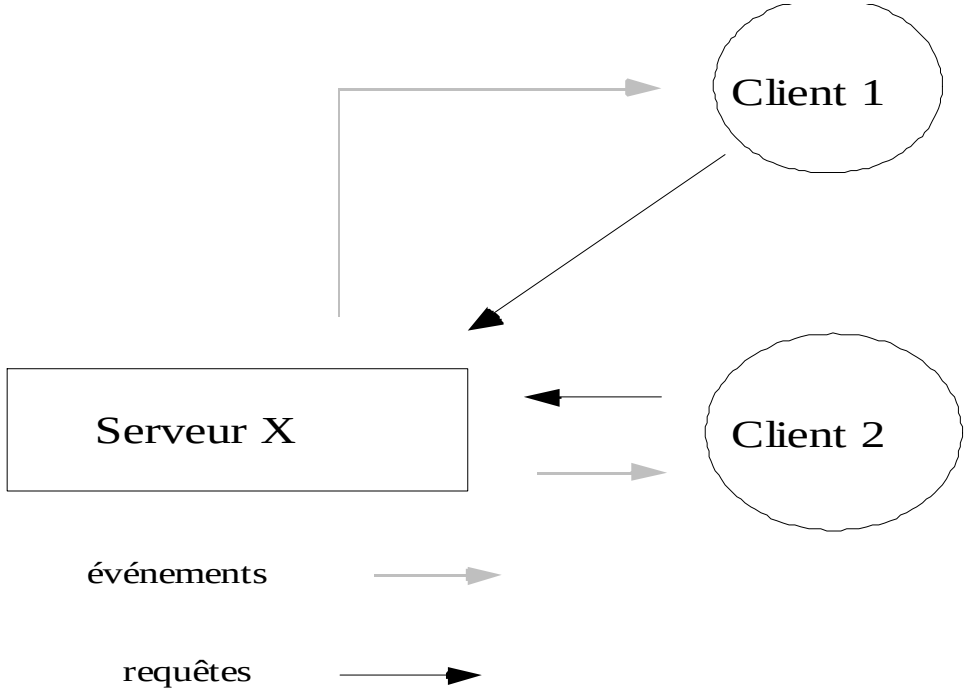
peuvent s'exécuter sur la même machine que le serveur ou sur une machine distante

- communiquent avec le serveur par le protocole X
- utilisent la librairie Xlib

Communication client-serveur

- client → serveur : requêtes d'affichage
  - messages de taille réduite → allégement trafic
  - optimisation au niveau du serveur X par le partage de ressources communes entre clients: polices de caractères, tables de couleurs, etc.
- serveur → client : événements
  - clavier, souris, demande de réaffichage (expose)
- les clients ne communiquent directement pas entre eux

Exemple de dialogue X



Le serveur X:

- distribue les entrées (clavier-souris) de l'utilisateur (converties en événements) aux différents clients (applications)
- reçoit les requêtes d'affichage des différents clients et les dessine à l'écran.

Exemple :

L'utilisateur tape des caractères dans une fenêtre, le serveur envoie ces caractères au client concerné (associé à cette fenêtre). C'est au client à décider que ces caractères ont un écho (ou pas)

Le protocole X

- définit le format des messages, le type des messages échangés et les règles de communication.
- indépendance du système d'exploitation, du matériel.
- rend transparents les protocoles de communications de bas niveau

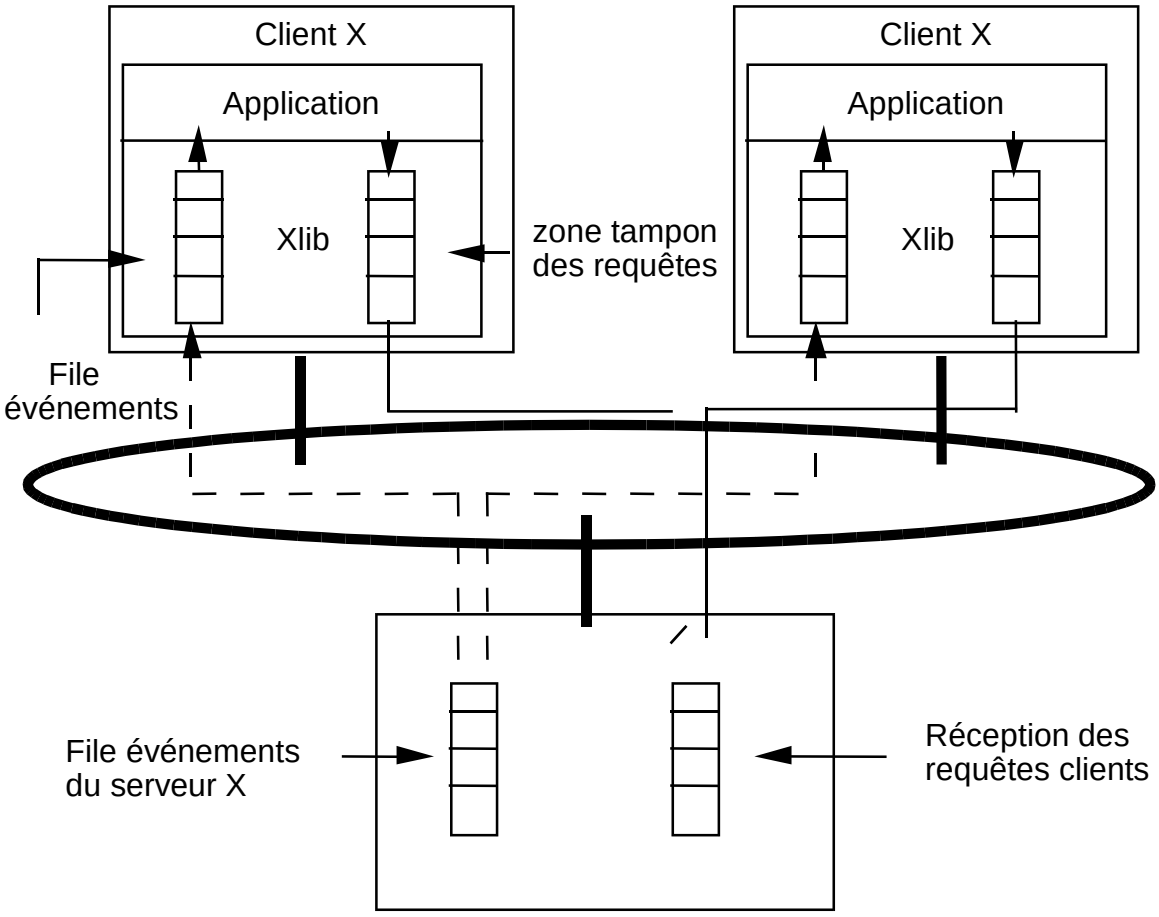
Caractéristiques :

- asynchrone, full duplex
- pas de confirmation de réception de messages.
- les requêtes clients ne sont en général pas envoyées directement mais sont stockés temporairement dans un tampon limitant ainsi le nombre d'accès réseau.

Types de messages

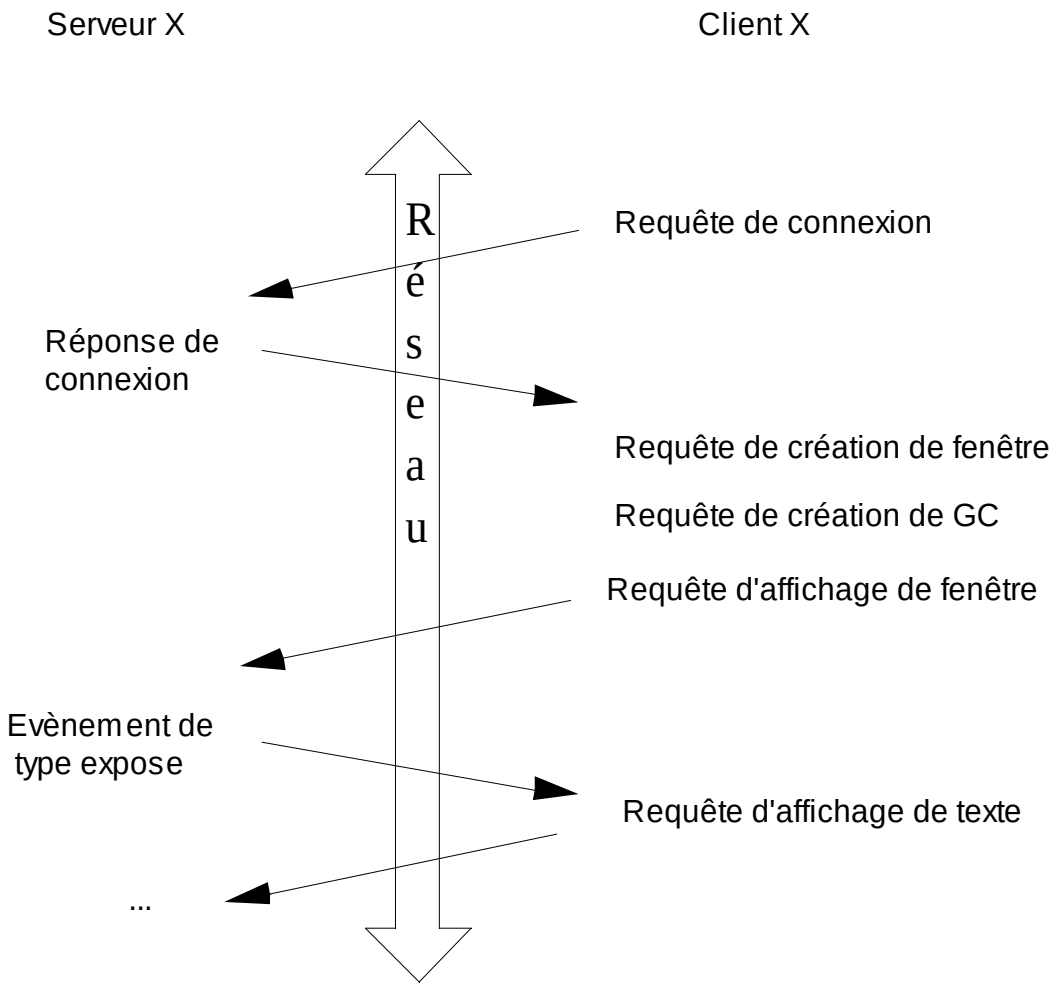
- Les requêtes à sens unique (client X -> serveur X)
  ex : Création de ressources X
- Les requêtes avec réponse (client X -> serveur X)
  ex : Information sur les ressources X créées auparavant
- Les événements (serveur X -> client X)
  ex : Pression d'un bouton de la souris
- Les réponses (serveur X -> client X)
  ex : taille d'une fenêtre
- Les messages d'erreurs
  ex : connexion non établie

Transmission de messages X protocolaires



X = protocole asynchrone

Les requêtes ne sont pas forcément envoyées immédiatement :



Window Manager

= client particulier qui permet de
- retailler les fenêtres
- réarranger les fenêtres sur l'écran
- convertir des fenêtres en icônes
- faire apparaître la fenêtre en premier/arrière plan
- détruire une fenêtre

Exemple : `mwm, twm`

En général, il "habille" les fenêtres d'un cadre. Le cadre autour des fenêtres n'est donc pas mis par les applications, ni par le protocole X mais par le WM.

Le WM impose l'aspect et la manière de manipuler les fenêtres : le "look and feel"

Exemple d'échange X avec WM : retaillage d'une fenêtre « xeyes »

1°) l'utilisateur appuie sur la souris au bord de la fenêtre et déplace la souris : "interruption" envoyée au serveur
2°) Le serveur reçoit cette interruption, la "transforme" en événement et l'envoie au WM
3°) Le WM retaille la fenêtre, réajuste son contour, ... et envoie des requêtes au serveur pour d'autres clients indirectement concernés par ce retaillage (masquage ou démasquage de leurs fenêtres)
4°) Le serveur X informe (= envoie des événements) tous les clients concernés entre autre `xeyes`
5°) `xeyes` reçoit cet événement et réajuste son dessin

Remarques

Quand on lance un client sur une machine distante, il faut avoir indiqué sur quelle machine et écran vont se faire les entrées sorties. En général cela est précisé par la variable `DISPLAY`. Par exemple, lancer sur la machine distante, la commande (en csh)

```
% setenv DISPLAY NomMachineLOCALE:0.0
```

Il faut de plus que la machine distante soit autorisée à afficher sur la machine locale. Ceci est obtenue à l'aide de la commande `xhost` lancée sur la machine locale. `xhost` permet d'ajouter ou de retirer des machines autorisées à afficher leur sorties sur la machine locale.

Exemples :

`% xhost +` (toutes les machines peuvent afficher)

`% xhost -` (seules peuvent afficher les machines autorisées (dans `/etc/X<Nom_serveur_X>.hosts`)

`% xhost -NomMachine` (NomMachine ne peut plus afficher)

`% xhost +NomMachine` (NomMachine peut afficher)

Conclusion

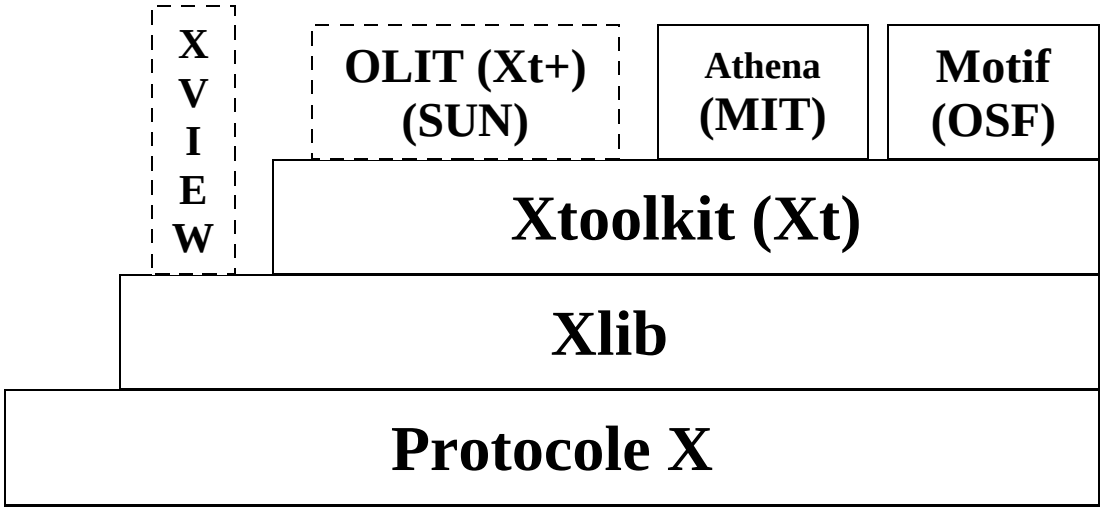
Sur la machine DISTANTE taper

```
% setenv DISPLAY NomMachineLOCALE:0.0
```

Sur la machine LOCALE taper

```
% xhost +
```

La « pile » X Window



Structure d'un programme Xlib

Inclusion de fichiers .h X11

Déclarations (globales) de structure de données nécessaires à notre application (Window, Display,...)

la fonction main() contient :

- une connexion au serveur X
- le numéro de l'écran où se font les affichages
- créer des fenêtres
- mapping des fenêtres
- sélection des événements à traiter
- la boucle d'événements
- libération des ressources allouées par le programme

Exemple de programme

Ce programme crée une fenêtre. Quand on appuie sur un bouton de la souris, l'application se termine.

```
#include <stdio.h>
#include <X11/Xlib.h>

#define VRAI 1
#define FAUX 0

int     fini;

main(argc, argv)
int argc;
char *argv[ ] ;
{
Display *mon_display;
Window  ma_window;
XEvent  evenement;
int     mon_ecran;
unsigned long couleur_bord, couleur_fond_fen;
void    erreur();
void    TraiteEvenement();

/* Les initialisations necessaires */
  if ((mon_display = XOpenDisplay(NULL)) == NULL)
                erreur("XOpenDisplay","");
mon_ecran = DefaultScreen(mon_display);

/* Valeurs par default des pixels */
        couleur_fond_fen = WhitePixel (mon_display, mon_ecran);
        couleur_bord = BlackPixel(mon_display, mon_ecran);

/* Creation de la fenêtre */
        ma_window = XCreateSimpleWindow (mon_display,
                                DefaultRootWindow (mon_display),
                                200, 300, 350, 250,
                                5,   /* Bordure de 5 pixels */
                                couleur_bord,
                                 couleur_fond_fen);
```

```
/* Selection des evenements a gerer */
        XSelectInput (mon_display, ma_window,
                ButtonPressMask /* evenement de bouton de souris */
                );

/* Enfin l'affichage de la fenêtre. Avant on l'avait seulement créée.
 * Lorsque le serveur aura traite XMapWindow, il retourne un
 * evenement d'exposition, evenement qui sera exploite par
 * la boucle d'evenements */

                XMapWindow (mon_display, ma_window);

/* LA BOUCLE D' EVENEMENTS */
        fini = FAUX;
        while (! fini) {
                XNextEvent (mon_display, &evenement);
                TraiteEvenement(&evenement);
          }

/* Terminaison correcte de cette monumentale application */
                XDestroyWindow (mon_display, ma_window);
                XCloseDisplay (mon_display);
}
void TraiteEvenement(pt_Evt)
XEvent *pt_Evt;
{  switch (pt_Evt->type){
                case ButtonPress:
/* Quand un bouton est enfonce, fini passe a vrai ce qui terminera
l'application. OK ?*/
                        fini = VRAI;
                 break;
                } /* switch */
} /* TraiteEvenement */

void erreur(s,t)
char *s, *t;
{ printf("Erreur : %s %s\n",s,t); exit(1);}
```

Pour compiler un programme Xlib mis dans le fichier `first.c` :

```
% cc first.c -lX11 -o first
```

La boucle d'événements

événements = interruptions générées par certaines manipulations (entrées clavier, reconfiguration des fenêtres, appui de bouton souris par l'utilisateur, ...)

- les événements sont rangés dans une file
- le programme lit les événements dans cette file

```
XEvent  EvtATraiter;
while (1) {
   XNextEvent(display, &EvtATraiter);
   switch (EvtATraiter.type) {
      case Expose : /* Traitement des événements d'exposition */

      break;
      case ButtonPress : /* Traitement des événements bouton
souris */
      break;
      case KeyPress :  /* Traitement des événements clavier */
      break;
      ....
}
```

Ces interruptions sont structurées : structure *XEvent* (lieu sur l'écran où a été émis le clic souris, date de retaillage d'une fenêtre, ...)

Les événements reçus par une fenêtre ont été envoyés par le serveur X.

=> le serveur X a donc fait un travail de tri : quel programme doit recevoir quels événements.

=> chaque fenêtre d'un programme X doit informer le serveur quels sont les événements qui l'intéressent.

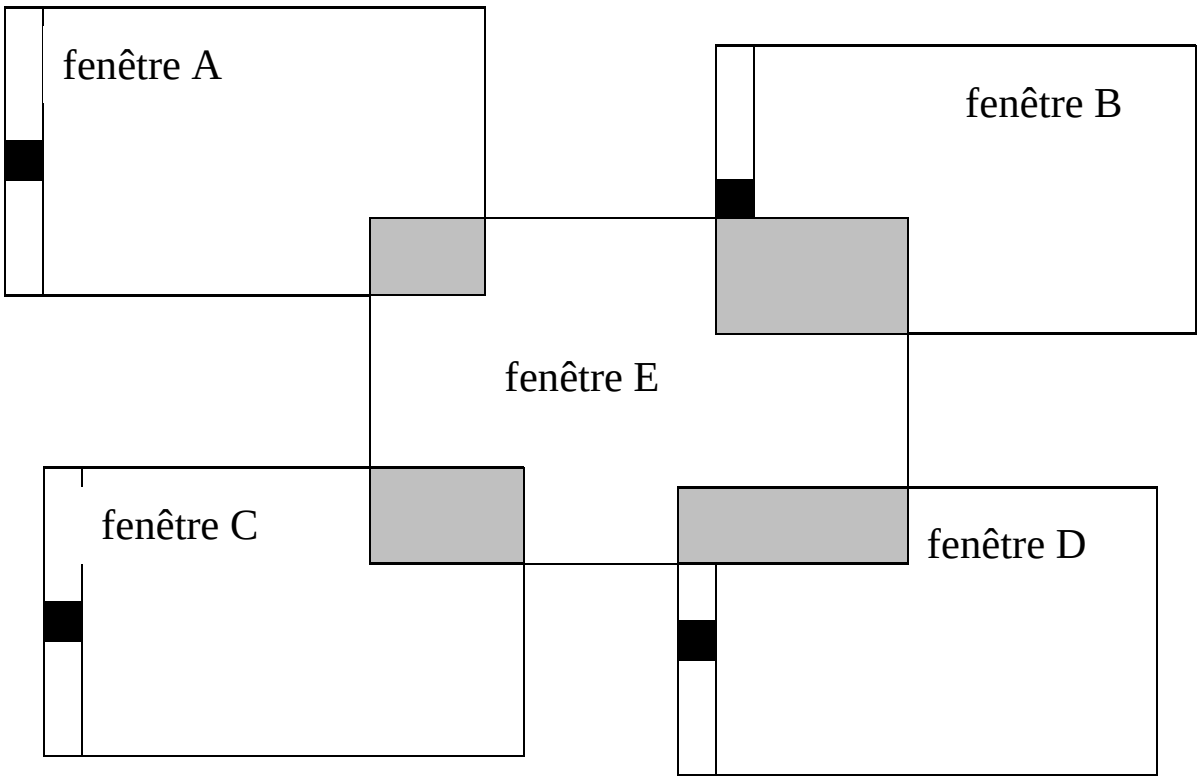
XSelectInput(display, Fen, MasqueEvt)

L'événement Expose

Généré lorsqu'une fenêtre est mappée ou devient visible après avoir été cachée par une autre (même en partie).

C'est généralement à la charge de l'application de redessiner sa fenêtre, pas au serveur X.

Lorsqu'une fenêtre en arrière plan passe au premier plan, elle doit redessiner certaines de ces parties. Le serveur X lui a envoyé pour cela éventuellement plusieurs Expose Event.



Quand la fenêtre E passe en premier plan, elle reçoit des événements Expose pour chaque zone d'intersection.

Contexte graphique

Pour dessiner dans les fenêtres, la Xlib offre des fonctions graphiques. Pour transmettre les paramètres graphiques à ces fonctions (couleur, épaisseur des traits, police, style de remplissage des surfaces, etc.), on utilise des trousses de dessin.

Contexte graphique (GC) = un identificateur de trousse de dessin

Avantages :

- on évite d'avoir à passer un nombre trop important d'arguments dans les fonctions graphiques.
- performance réseau : la trousse de dessin est créée et gardée par le serveur, et repérée dans les requêtes du client par son ID (le GC). Un GC peut être partagé par deux clients distincts.
- on peut créer plusieurs GC pour un client et indiquer à chaque tracé quel GC on utilise.

Création de contexte graphique

```
GC XCreateGC(display, drawable,
                masque, AdrValeursGC)
```

- `AdrValeursGC =` adresse d'une structure qui contient les paramètres graphiques
- `masque` = identifie les paramètres utilisés

Exemple d'utilisation

```
GC gc;
XGCValues ValeursGC;
…
ValeursGC.foreground = BlackPixel(display, screen);
ValeursGC.line_width = 4;
```

```
gc = XCreateGC(display, RootWindow(display, screen),
                        GCForeground | GCLineWidth, &ValeursGC);

/* On peut désormais utiliser ce GC */
```

Une variante plus lisible :

```
GC gc;
…
gc = XCreateGC(display, RootWindow(display, screen), 0, NULL);
XSetForeground(display, gc, BlackPixel(display, screen));
XSetLineWidth(display, gc, 4);

/* On peut désormais utiliser ce GC */
```

La structure `XGCValues`

Les principaux champs :

- `foreground` : couleur du trait
- `background` : couleur de fond
- `font` : police de caractères
- `line_width` : épaisseur de trait
- `line_style` : style de trait
- `fill_style` : style de remplissage d'une zone
- `function` : fonction de dessin

La Couleur

La plupart des écrans actuels sont gérés à l'aide d'une table de couleurs (colormap, look-up table, …).

Un point à l'écran (pixel) détermine sa couleur à l'aide d'un indice dans la table de couleurs. A cet indice dans la table se trouvent les caractéristiques de la couleur (quantité de rouge, vert, bleu). On a donc :

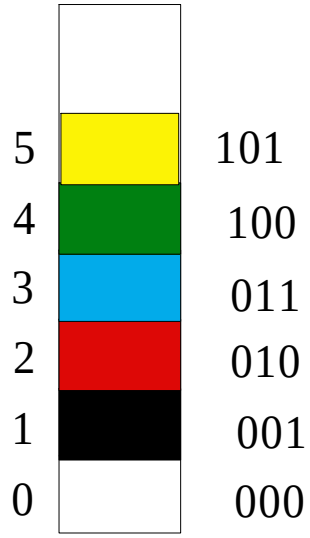


Table de couleurs 1

Changer le contenu d'une entrée signifie que tous les dessins qui l'utilisent changent de couleur.

Certaines applications utilisent leur propre table de couleurs. Quand le curseur pénètre dans leur espace => flash assez désagréable, car toutes les autres couleurs changent

Le champ `function` du GC

Le champ `function` du contexte graphique détermine la manière de combiner la couleur de dessin avec celle sur laquelle on dessine.

Entre le pixel `dst` (déjà dessiné) et le pixel qu'on veut dessiner (dit pixel source `src`) on fait l'opération logique: `src function dst`

Remarque : pixel = indice dans la table de couleurs !
function = opération logique bit par bit

Mise à jour du champ `function`

```
XSetFunction(display, gc, fonction)
```

Fonction logique	Définition
GXclear	0
GXand	src AND dst
GXandReverse	src AND (NOT dst)
GXcopy	src
GXandInverted	(NOT src) AND dst
GXnoop	dst
GXxor	src XOR dst
GXor	src OR dst
GXnor	(NOT src) AND (NOT dst)
GXequiv	(NOT src) XOR dst
GXinvert	NOT dst
GXorReverse	src OR (NOT dst)
GXcopyInverted	NOT src
GXorInverted	(NOT src) OR dst
GXnand	(NOT src) OR (NOT dst)
GXset	1

Les fonctions les plus utilisées

- `GXcopy` : copie le pixel `src` en ignorant `dst`
- `GXxor` : une seconde application efface le premier dessin (utile pour les animations)

Le champ `function` du GC (suite)

Le résultat des couleur obtenue sera une opération mathématique bit à bit.
Ce n'est pas du dessin d'art !

Ou trouver X Window ?

X Window et windows

Une implantation gratuite de X Window et disponible à partir de `http://x.cygwin.com/` : le projet Cygwin/X.
Un excellent manuel d'utilisation et d'installation décrit le produit.
Un serveur X peut être lancé sur l'écran windows par `startxwin.bat`
(`REP_INSTAL\usr\X11R6\bin\startxwin.bat`)
Nécessite d'avoir installé le projet cygwin (API Unix pour windows).

X Window et Macintosh

Disponible à :
`http://www.apple.com/downloads/macosx/unix_open_source/xfree86.html`